PRONNUS PROLOG

ARQUITETURA E IMPLEMENTAÇÃO

Valério Machado Dallolio

Luiz Fernando Pereira de Souza

NCE-01/89

Grupo de Inteligência Artificial/UFRJ

março/89

Universidade Federal do Rio de Janeiro

Núcleo de Computação Eletrônica

Caixa Postal 2324

20001 - Rio de Janeiro - RJ

BRASIL

Este relatório foi parcialmente financiado com recursos do Projeto ESTRA da SID Informática S.A.

ÍNDICE

SINOPSE

Este relatório descreve a arquitetura e implementação do Interpretador Pronnus Prolog. São abordados os aspectos relevantes que diferenciam a implementação do Prolog das linguagens tradicionais, tais como análise voltada para expressões, geração de código baseado em pseudo-instruções, gerência de código em tempo de execução e interpretação do código gerado.

ABSTRACT

This report describes the Pronnus Prolog Interpreter's architecture and implementation. The remarkable aspects that differentiate the Prolog implementation from traditional languages, such as analysis with respect to expressions, code generation based on pseudo-intructions, code management at execution time and interpretation of the generated code, are also broach.

1. INTRODUÇXO

O Grupo de Inteligência Artificial da Universidade Federal do Rio de Janeiro vem, desde o segundo semestre de 1987, pesquisando sobre a implementação da linguagem Prolog. Para isto foi criado dentro do Núcleo de Computação Eletrônica da UFRJ o Projeto Prolog, cuja meta inicial é desenvolver um ambiente para esta linguagem sobre sistemas operacionais tipo Unix.

Com este projeto será dominada a tecnologia envolvida na implementação de um interpretador Prolog (o Pronnus Prolog) e a partir deste conhecimento poderão ser realizadas pesquisas nas áreas de arquiteturas dedicadas à Prolog e de extensões a esta linguagem. Além disto, não encontra-se disponível no Brasil nenhum interpretador Prolog para ambientes tipo Unix. Desta forma, os aplicativos do Grupo de Inteligência Artificial da UFRJ, que fossem desenvolvidos nesta linguagem, não teriam acesso às famílias de computadores baseados neste sistema operacional.

Este relatório apresenta a arquitetura do interpretador Pronnus Prolog, em desenvolvimento, abordando vários problemas inerentes à implementação da linguagem Prolog e apresentando as soluções adotadas. São vistos em particular os módulos de análise, compilação e execução. Finalizamos discutindo algumas extensões a serem feitas na linguagem.

2. HISTÓRICO

Definida por Alain Colmerauer no início da década de setenta, Prolog se caracteriza por tentar ser uma implementação da lógica de primeira ordem. Embora este objetivo não tenha sido plenamente alcançado, a linguagem mostrou-se apropriada para a programação não-procedimental, principalmente de aplicativos de inteligência artificial, como por exemplo sistemas especialistas. Entretanto, Prolog não deixa de ser uma linguagem de uso geral.

A primeira implementação eficiente da linguagem foi feita pela equipe de David Warren. Esta implementação era baseada na geração de código para uma máquina abstrata. Em 1983, Warren definiu um conjunto de pseudo-instruções para uma segunda máquina abstrata [Warren 83]. Esta nova arquitetura apresentava uma sensível evolução em relação à primeira.

A implementação do interpretador Pronnus Prolog é baseada na geração de um código intermediário, que é uma modificação das pseudo-instruções da segunda máquina de Warren [Souza 88a, Souza 88b, Dallolio 88].

O Projeto Prolog começou com o estudo e a definição das pseudo-instruções e do ambiente de execução a serem utilizados [Souza 88b]. A fim de validar a arquitetura, foi implementado um simulador do ambiente de execução. Feito isto, implementamos um gerador de código, que traduz cláusulas Prolog para a linguagem de montagem de nossa máquina abstrata.

Atualmente, estamos trabalhando na implementação da análise sintática, do núcleo do interpretador e da interface com o usuário. À medida que integrarmos aos módulos em desenvolvimento o gerador de código, o simulador e implementarmos os predefinidos, teremos uma primeira versão do interpretador Pronnus Prolog.

3. ARQUITETURA

Um programa Prolog contitui-se de uma coleção de predicados. Estes se dividem em predefinidos, que correspondem à biblioteca de funções da linguagem, e predicados programados pelo usuário. Cada predicado é composto por uma seqüência de cláusulas e se caracteriza por um funcional, o nome do predicado, e uma aridade, que é o número de parâmetros. Eles são referenciados pela notação "funcional/aridade".

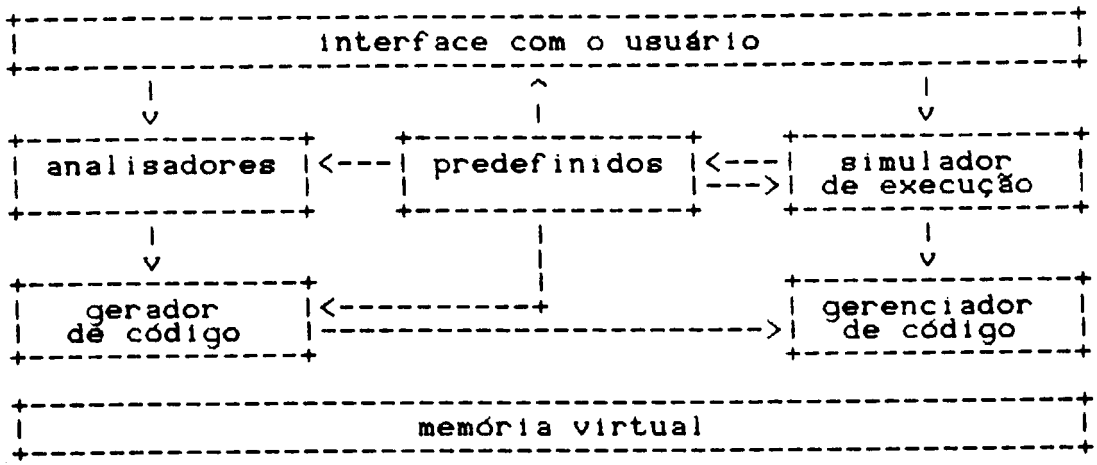
Uma cláusula é composta por uma cabeça, com o funcional e a aridade do predicado a qual pertence, e por um corpo, que é uma seqüência de "chamadas" a outros predicados. No exemplo abaixo, uma cláusula do predicado "avo/2" possui duas "chamadas" ao predicado "pai/2":

```
avo( Avo, Neto ) :- pai( Avo, X ), pai( X, Neto ).
```

A unidade de um programa Prolog são as cláusulas. Estas são, em sua maior parte, pequenas como no exemplo. Isto, aliado à característica interativa de execução da linguagem e à possibilidade de alteração do código de um programa Prolog em tempo de execução, fazem com que o tipo de ambiente mais adequado a esta linguagem seja um interpretador.

O Pronnus Prolog se apresenta para o usuário como um interpretador. Entretanto, internamente, as cláusulas são compiladas para uma representação intermediária baseada em pseudo-instruções.

A arquitetura do interpretador pode ser representada pelo diagrama abaixo:



A medida que a cláusula é lida, ela é reconhecida pelos módulos de análise gerando uma representação interna. A partir desta representação, o módulo de geração de código constrói uma seqüência de pseudo-instruções que implementam a semântica da cláusula. Finalmente, o módulo de gerenciamento de código incorpora a cláusula ao predicado correspondente. Em tempo de execução, o módulo de simulação executa a semântica associada a cada pseudo-instrução. Estes módulos serão vistos com algum detalhe ao longo deste texto.

4. INTERFACE

O módulo de interface, além de executar a troca de informações entre o usuário e o interpretador, é responsável pela manipulação dos arquivos do usuário. Hoje em dia, é consenso geral que para um programa ser considerado bom, não basta que ele funcione corretamente, é indispensável uma boa interface com o usuário e um bom manual. O Pronnus Prolog vem sendo desenvolvido segundo esta orientação.

Existem quatro pontos na interface com o usuário do Pronnus Prolog que merecem destaque. Primeiramente, são aceitos como válidos todos os caracteres acentuados, permitindo desta forma a utilização de comentários, átomos e constantes acentuados.

Em segundo lugar, na entrada de dados via teclado foi elaborado um esquema que permite que o usuário tenha conhecimento do que o sistema está esperando como entrada, evitando situações que ocorrem em outros interpretadores, onde o usuário e o sistema ficam um esperando o outro. Isto é feito através da colocação de "prompts" diferenciados, de forma que o usuário saiba se o sistema espera uma nova cláusula, a continuação de uma cláusula iniciada anteriormente ou pelo fim de um comentário.

Em terceiro lugar, ao contrário de outros interpretadores onde as cláusulas listadas tem como nome das variáveis um padrão de numeração, o Pronnus Prolog guarda o nome originalmente dado à cada variável, e exibe as cláusulas utilizando este nome. Isto é muito útil, principalmente na depuração de programas.

Por último, vale mencionar que toda as mensagens produzidas pelo Pronnus Prolog podem ser personalizadas para cada usuário, permitindo que sejam impressas mensagens na língua mais conveniente ao usuário.

5. ANÁLISE

A linguagem Prolog não só apresenta uma forma de execução bem diferente, mas a sua própria sintaxe é bastante peculiar. A sintaxe do Prolog pode ser definida como um pequeno conjunto de elementos básicos, que podem ser combinados através de operadores formando expressões. O usuário pode redefinir ou remover operadores já existentes e acrescentar novos.

Vejamos alguns dos elementos básicos da linguagem (onde "exp" denota uma expressão qualquer):

```
Variável            -> uma variável,
átomo               -> um átomo,
´Um átomo´          -> um átomo,
#$:                 -> um átomo,
fun( exp, exp )     -> uma estrutura,
[ exp | exp ]       -> uma lista e
[ exp, exp, exp ]   -> uma lista.
```

Uma expressão, ao ser lida pela análise, é entendida como uma estrutura, onde o operador é o funcional e os operandos são os argumentos. Esta transformação é guiada pelo reconhecimento de operadores dentro da expressão. Abaixo temos um exemplo de uma expressão e sua representação como estrutura. No caso, funcionam como operadores os átomos: ":-", "," e ";". A estrutura do exemplo poderia estar representando uma cláusula, dependendo do contexto.

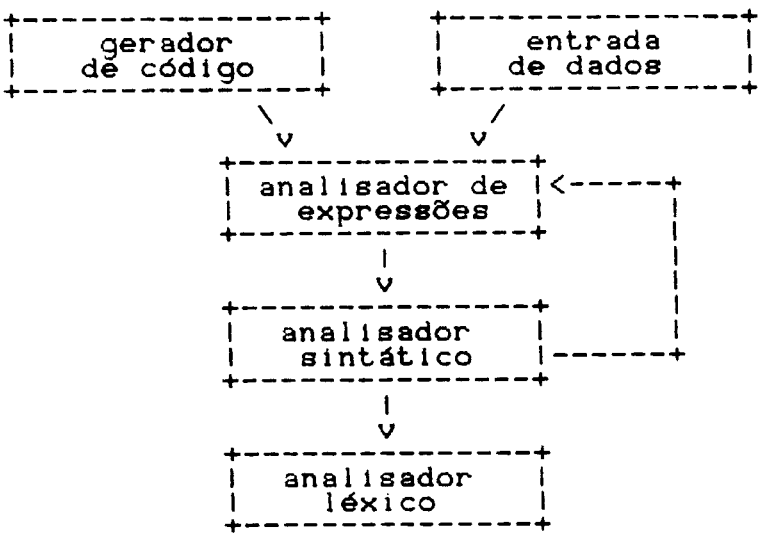
Exemplo:

```
Expressão:  f :- a , ( b ; c )
Estrutura:  ´:-´( f, ´,´( a, ´;´( b, c )))
```

Um operador pode ser redefinido em tempo de execução, alterando as suas propriedades (veremos adiante), de forma que ele assuma uma das muitas combinações possíveis. Logo, a estrutura de análise e reconhecimento da linguagem Prolog não se encaixa completamente no esquema

tradicional, uma vez que a maioria dos elementos da linguagem, operandos e operadores, não são fixos.

No Pronnus Prolog, a análise foi hierarquizada em três níveis, conforme o diagrama abaixo:

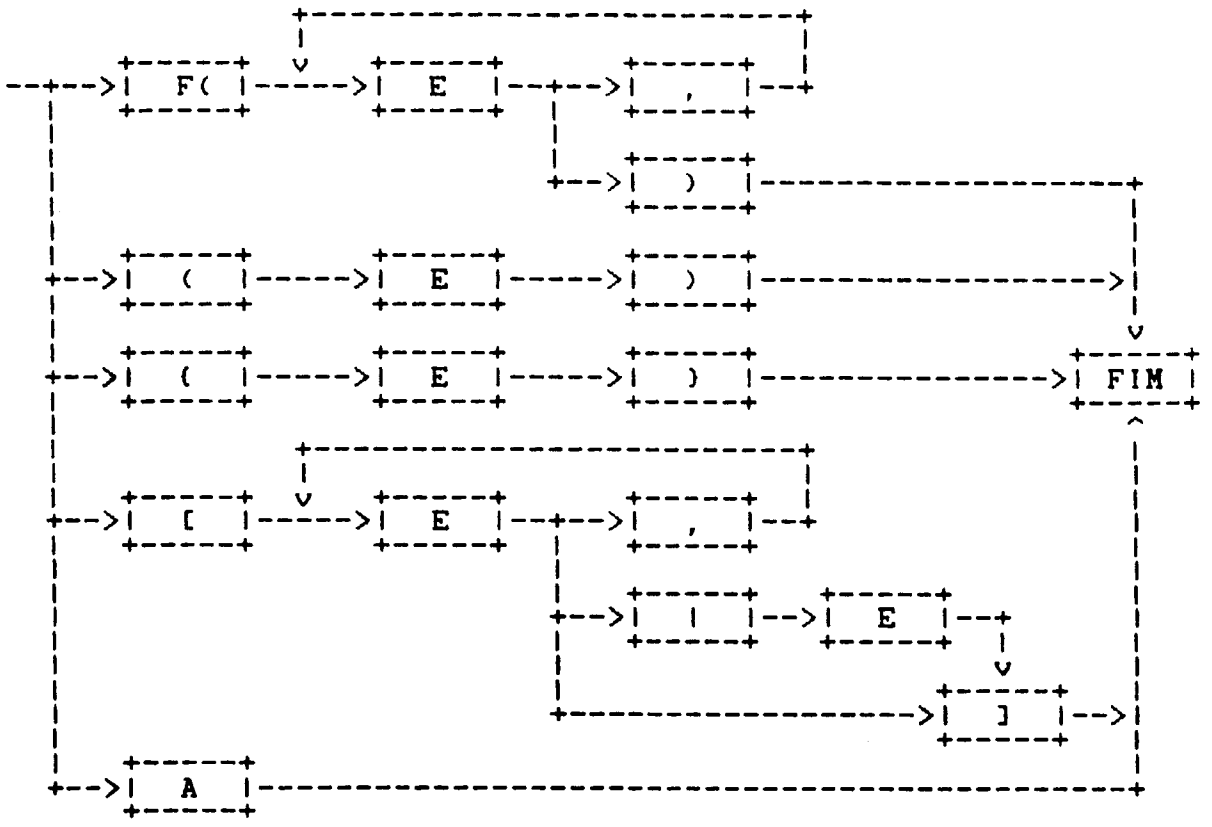


Os analisadores léxico e sintático realizam o reconhecimento dos elementos básicos da linguagem. Já o analisador de expressões agrupa estes elementos de acordo com as precedências e associatividades de cada operador.

Pode ser notado que a própria hierarquia entre os módulos é reflexo da estrutura sintática da linguagem. Uma expressão pode conter uma estrutura que contém expressões; logo, o analisador de expressões chama o sintático, que por sua vez chama uma nova instância do analisador de expressões.

5.1. ANALISADOR SINTÁTICO

No analisador sintático procuramos deixar a parte fixa da linguagem, ou seja, a estrutura dos elementos componentes (listas, estruturas, átomos). Desta forma, chegamos ao seguinte digrafo sintático:



Onde "E" denota uma expressão, que causa uma chamada ao analisador de expressões, "A" denota um átomo qualquer, que pode ser tanto um operador como um operando, e "F(" denota um funcional.

5.2. SINTAXE DE OPERADORES

Os operadores representam na verdade uma forma sintaticamente mais conveniente de se escrever certas estruturas. Por exemplo, expressões aritméticas são comumente escritas através do uso de operadores. Para o Prolog, uma expressão do tipo "X + Y * Z" traduz-se na estrutura "+(X, *(Y, Z))", se mantidas as propriedades

inicialmente definidas para os operadores "+" e "*" (ver Apêndice - Tabela de Operadores).

É importante notar que os operadores não ativam nenhum tipo de aritmética, de modo que, em Prolog, "3 + 4" não significa o mesmo que "7". Para que uma estrutura seja entendida e avaliada como uma expressão aritmética, esta deve ser submetida ao predicado de avaliação "is/2" [Clocksin 87].

Cada operador possui três propriedades: posição, classe de precedência e associatividade. A posição pode ser infixa, prefixa ou posfixa. A classe de precedência é um inteiro na faixa de 1 a 1200 que nos indica que operação deve ser primeiramente executada. Tanto maior é a precedência do operador quanto mais próximo de 1 é a sua classe de precedência. Por último, a associatividade é utilizada para resolver questões de ambigüidade em expressões que possuam dois ou mais operadores com a mesma precedência.

Associamos a cada operador um átomo especial, especificando sua posição e associatividade, conforme a tabela abaixo:

xf	Operador posfixo não associativo
yf	Operador posfixo associativo
xfx	Operador infixo não associativo
yfx	Operador infixo associativo à esquerda
xfy	Operador infixo associativo à direita
fx	Operador prefixo não associativo
fy	Operador prefixo associativo

Vale ressaltar o seguinte detalhe: um mesmo átomo pode representar operadores com características completamente diferentes, cabendo a análise decidir, de acordo com a expressão que está sendo analisada, qual deles aplicar. Um caso conhecido e bastante trivial é o do átomo "-" que tanto pode estar representando o operador yfx, 500

(menos infixo, associativo a esquerda e precedência 500), como também o operador fx, 500 (menos prefixo, não associativo e precedência 500). Além disso, qualquer operador cujas propriedades não se "encaixem" adequadamente na expressão que o contém, passa a ser visto como um operando, e não mais como um operador.

5.3. ANALISADOR DE EXPRESSÕES

O funcionamento básico do analisador de expressões consiste em agrupar elementos fornecidos pelo sintático em uma estrutura. Utiliza-se para isto de uma pilha de expressões, onde são temporariamente armazenados operandos e operadores, até que estes possam ser reunidos em estruturas intermediárias, que são também guardadas na pilha e vistas como operandos.

Cada elemento retornado pelo sintático pode ser um operando (inteiro, real, átomo, lista ou estrutura) ou um átomo representando um operador. No caso de operador, o analisador de expressões decide dentre as diversas formas com que este encontra-se definido, aquela que deverá usar. Eventualmente o operador poderá ser usado até como um operando.

Para tomar esta decisão, o analisador de expressões realiza os seguintes procedimentos:

a) Empilha o elemento corrente na pilha de expressões. Este passa a se chamar "topo".
b) Recebe o próximo elemento retornado pelo sintático. Este passa a se chamar "próximo".
c) Faz a compatibilização entre o "topo" e o "próximo", conforme será visto mais adiante.
d) Verifica se pode agrupar elementos da pilha de expressões em sub-estruturas.
e) O elemento corrente passa a ser o "próximo".
f) Caso o elemento corrente seja diferente de "fim de expressão" retorna ao procedimento "a".
g) Monta a estrutura final com os elementos que

estão na pilha e empilha esta estrutura.

A "compatibilização" mencionada no procedimento "c" decide as propriedades do "topo" e restringe as do "próximo", sendo dirigida pela tabela abaixo. Nesta tabela as linhas são indexadas pelas propriedades do "próximo" e as colunas pelas propriedades do "topo". A notação "a" significa "x" e/ou "y" (por exemplo, "fa" equivale a "fx" e/ou "fy") e "n" significa operando.

Cada posição da tabela está dividida em duas partes. A parte de cima, contém as propriedades que devem ser eliminadas do "topo", a fim de se determinar suas propriedades corretas para a expressão. Analogamente, a parte de baixo contém as propriedades que devem ser eliminadas do "próximo", pois estas são imcompatíveis com o "topo". Finalmente, algumas posições correspodem a situações de erro, e nesse caso a análise deve ser suspensa dando lugar a um procedimento de recuperação.

Topo / Próximo	n ou af ou n e af	fa ou afa ou fa e afa	n ou fa ou n e fa	af ou afa ou af e afa
n ou fa ou n e fa	Erro		n	af
	Erro			
af ou afa ou af e afa		Erro	fa	afa
		Erro		
demais combinações			n	afa
	n e fa	af e afa	af e afa	n e fa

Como saída, o analisador de expressões devolve um átomo ou uma estrutura. Este objeto é armazenado diretamente em uma das pilhas de execução do Prolog, a pilha global, de uma forma parecida com a que são armazenados os objetos da linguagem a tempo de execução. A diferença básica consiste

na presença de informações relativas às variáveis. No caso de uma expressão devolvida a uma operação de entrada e saída, estas informações não são utilizadas. Entretanto, se esta expressão está sendo devolvida ao gerador de código, estas informações serão úteis. Veremos na próxima seção um pouco sobre o gerador de código.

6. COMPILAÇÃO

A compilação de cláusulas Prolog em pseudo-instruções não é uma tarefa demasiadamente complexa. Primeiramente, porque as pseudo-instruções foram definidas única e exclusivamente para suportar esta linguagem, sendo quase biunívoca a correspondência entre os símbolos da linguagem e as instruções geradas. Em segundo lugar, porque os mecanismos de execução do Prolog (unificação e retrocesso [Souza 88b]), embora muito poderosos, são bastante simples. Entretanto, devido a presença de caracteríticas particulares do Prolog, completamente diferentes das tradicionalmente encontradas em outras linguagens de programação, este processo de tradução traz aspectos relevantes.

6.1. ANÁLISE DAS VARIÁVEIS

Se as cláusulas Prolog não possuíssem variáveis, a geração de código seria extremamente fácil. Podemos dizer que todo o processo de compilação está centrado na análise das variáveis.

Do ponto de vista de implementação, as variáveis em Prolog são de dois tipos: temporárias e permanentes. Variáveis temporárias são alocadas em registradores, comuns aos predicados, não sendo portanto preservadas entre a execução de dois objetivos (ou duas "chamadas"). Já as permanentes são alocadas na pilha de execução (pilha local), mais precisamente no registro de ativação (ambiente) da cláusula em questão, e continuam existindo mesmo entre chamadas a procedimentos.

Dentro de uma cláusula, variáveis podem ser obtidas de duas formas: através da passagem de parâmetros (externa a cláusula ou instanciada), ou criadas dentro da própria cláusula. Variáveis são criadas no momento em que aparecem pela primeira vez, localmente à cláusula (na pilha local ou em registradores), ou em uma área de alocação dinâmica (pilha global [Souza 88b]). Além disso, algumas instruções

alteram a alocação das variáveis, fazendo, por exemplo, com que uma variável alocada na pilha local passe para a pilha global.

Fica claro portanto que para podermos gerar o código, precisamos conhecer um atributo, que chamamos de estado, associado a cada váriavel. Este estado marca a forma de alocação da variável e o seu uso até aquele momento (se já foi criada, onde está alocada, se ainda vai ser utilizada). A instrução a ser gerada depende do estado das variáveis envolvidas e esta instrução eventualmente altera aqueles estados. É importante ressaltar que este atributo só existe em tempo de compilação, embora seja absorvido de certa forma pela instrução gerada. Resumimos no quadro abaixo os possíveis estados de uma variável e o efeito das instruções sobre estes estados. Instruções que alteram o estado das variáveis estão indicadas com a seguinte notação: estado antigo --> estado novo. (Utilizamos a notação "T" para as variáveis temporárias e "P" para as permanentes.)

	GET	UNIFY	PUT
Estado 0: Não criada	Get_variable T 0 --> 1 Get_variable P 0 --> 1	Unify_variable T 0 --> 2 Unify_variable P 0 --> 2	Put_variable T 0 --> 2 Put_variable P 0 --> 3
Estado 1: Externa cláus.	Get_value T Get_value P	Unify_local T 1 --> 2 Unify_local P 1 --> 2	Put_value T Put_value P
Estado 2: Na Pilha Global	Get_value T Get_value P	Unify_value T Unify_value P	Put_value T Put_value P
Estado 3: Local a cláus.	Nunca Ocorre	Unify_local T 3 --> 2 Unify_local P 3 --> 2	Put_value P Put_unsafe P 2 --> 3

Instruções com prefixo "Put" são usadas na preparação de argumentos nas "chamadas" a predicados. Elas sempre tem como resultado o armazenamento de algum valor em um dos registradores de passagem de parâmetros [Souza 88b]. Instruções com prefixo "Get" são usadas no início de cada cláusula para obter os argumentos. Por último, as intruções com prefixo "Unify" são usadas para unificar os argumentos de estruturas ou listas.

6.2. GERAÇXO DO CóDIGO

O processo de compilação foi particionado em quatro etapas. Na primeira etapa, é recebida uma estrutura montada pelo analisador de expressões, que sofre um pré-processamento, onde são inicializadas algumas informações, como por exemplo se uma variável é temporária ou permanente.

Na segunda etapa, são geradas as pseudo-instruções, que implementam a cláusula Prolog, encadeadas em forma de digrafo acíclico. Nesta fase não nos preocupamos com alocação das variávies. Simplesmente coletamos as informações sobre a evolução de seus estados, levando em conta o efeito das instruções geradas. Assim, algumas instruções que dependem de informações relativas à alocação [Dallolio 88] ficam incompletas.

Na terceira etapa, é feita a alocação das variáveis. E finalmente, na última etapa, o código é copiado para uma área especialmente reservada, de forma seqüêncial, aplicando a alocação de variáveis, completando as instruçoões necessárias e realizando algumas otimizações locais. A área destinada a receber o código das instruções, assim como as áreas destinadas às três pilhas de execução do Prolog (local, global e trilha [Souza 88b]) são suportadas pelo módulo de gerência de memória. Este módulo foi desenvolvido com a finalidade de permitir que o usuário não ficasse restrito à memória principal em sistemas que não suportem memória virtual. Desta forma conseguimos alocar para cada uma destas áreas um mega-palavras (dois mega-octetos).

7. GERÊNCIA DE CÓDIGO

Um aspecto interessante na implementação da linguagem Prolog refere-se à dinâmica do código em tempo de execução. Existem certos predefinidos na linguagem ("call/1" e "not/1" [Clocksin 87]) que recebem uma estrutura como parâmetro e requerem que estas sejam compiladas e executadas. Isto acarreta que qualquer programa Prolog realmente compilado para uma máquina alvo, contenha em seu interior um gerador de código Prolog. Sem isto, a linguagem não estará completa.

Outra flexibilidade refere-se à introdução e remoção de cláusulas a tempo de execução ("assert/1" e "retract/1" [Clocksin 87]). Na execução do código de uma cláusula, uma "chamada" a outro predicado causa o desvio do fluxo de execução para as cláusulas deste predicado. Posteriormente retornamos à cláusula "chamadora". Entretanto, a linguagem permite que aquela cláusula tenha sido removida por um dos predicados "chamados". Ao realizarmos este teste em interpretadores comercializados no exterior para máquinas compatíveis com IBM-PC, constatamos que os mesmos, ou proíbem alteração de código a tempo de execução, descaracterizando a linguagem, ou apresentam erro de funcionamento nestas condições. No Pronnus Prolog, tivemos o cuidado de manter os códigos das cláusulas removidas intactos até o fim da execução de uma consulta. Desta forma, embora estas cláusulas não possam mais ser disparadas, qualquer retorno ainda pendente a elas não causará um mau funcionamento do interpretador.

O código de cada predicado pode ser hierarquizado em dois níveis: o código de cada cláusula e o código de seleção de cláusulas, que é responsável por ativar cada cláusula no momento apropriado [Souza 88b]. A introdução de uma nova cláusula causa a obsolescência do código de seleção. Logo, este precisa ser regenerado. Entretanto, este procedimento não é simples, visto que aquele código contém tabelas de espalhamento e outros mecanismos para acelerar a seleção da

cláusula correta. Desta forma, se o código do predicado for alterado a cada nova cláusula incluída, o processo de entrada de cláusulas será muito penalizado, principalmente na leitura de cláusulas de um arquivo. No Pronnus Prolog a inclusão de uma nova cláusula não causa a alteração imediata do código do predicado. Esta alteração é retardada o máximo possível, de forma que muita vezes fazemos uma só alteração para a inclusão de várias cláusulas.

As cláusulas, após serem compiladas, possuem duas formas de representação: a primeira, montada pela análise, em forma de estrutura e a segunda na forma de pseudo-código. A representação em forma de estrutura não deve ser descartada após a compilação, pois são utilizadas por dois predefinidos da linguagem: o "listing/0", encarregado de listar os predicados do usuário, e o "retract/1", encarregado de remover cláusulas. Sem ela, ambos os predicados passariam a envolver um processo de descompilação suficientemente complexo e demorado para invibializá-los.

8. EXECUÇXO

Os algoritmos e métodos utilizados na execução da linguagem Prolog são a parte mais sensível na tecnologia de implementação desta linguagem. Por isto, a primeira fase do Projeto Prolog foi o estudo da pouca bibliografia a respeito e a definição de um ambiente de execução para a linguagem, visto que os trabalhos analisados eram incompletos e omissos em pontos importantes.

Definido o nosso ambiente, não tínhamos certeza de ter coberto todos os pontos em aberto encontrados nos outros trabalhos, nem da correção das soluções adotadas. Desta forma, implementamos o simulador de nosso ambiente, que validou as nossas pseudo-instruções. Em vários testes realizados, apesar da quantidade de código de verificação e monitoração presentes no simulador, chegamos a uma performance de execução igual a dos melhores interpretadores.

O módulo de execução do Pronnus Prolog será construído a partir da adaptação do núcleo do simulador, visto que várias tabelas necessárias a execução do Prolog foram implementadas de forma simplificada. Será também necessária a programação dos predicados predefinidos, que embora não seja uma tarefa complexa, exige um grande esforço de programação.

9. PERSPECTIVAS

Apesar de todo o conhecimento que está sendo adquirido e divulgado a partir do desenvolvimento do interpretador Pronnus Prolog, este deve ser visto também como a pedra fundamental de várias frentes de pesquisa que estão surgindo, no NCE.

A primeira frente de pesquisa criada foi o desenvolvimento de uma arquitetura dedicada à execução de Prolog [Schmitz 88], que vem sendo feito por uma equipe formada por pesquisadores do Grupo de Projeto de Circuitos Integrados e do Grupo de Inteligência Artificial. O trabalho desta equipe se iniciou com a implementação de um co-processador Prolog baseado no simulador do ambiente de execução e tem como meta a elaboração de um circuito integrado dedicado à execução das pseudo-instruções Prolog.

Na parte de software as frentes que podem ser abertas se dividem em duas classes: convencionais e não convencionais. Nas aplicações convencionais temos: o desenvolvimento de interfaces com janelas e menus, gráficas e para "mouse", e interfaces com outras linguagens, sistemas operacionais e banco de dados convencionais [Berghel 85].

As frentes de pesquisa nas áreas não convencionais provavelmente darão tema para teses de mestrado e doutorado e para a criação de novos projetos. Dentre elas podemos prever: interface com banco de dados lógicos [Sciore 86], interface com banco de dados orientados a objetos, extensões para orientação a objetos [Banerjee 87], Prolog II e Prolog com processamento paralelo.

10. REFERÊNCIAS

[Warren 83] Warren D. H. D.,
An Abstract Instruction Set,
Technical Note 309, SRI International,
Menlo Park, California, outubro de 1983.

[Berghel 85] H. L. Berghel,
Simplified Integration of Prolog with RDBMS,
ACM Data Base,
New York, 1985.

[Sciore 86] Sciore E. e Warren D. S.,
Towards an Integrated Database-Prolog System,
Proceedings from First International Workshop,
Benjamin/Cumming Series in Data Systems and Applications,
pg 293 - 305,
Menlo Park, California, 1986.

[Clocksin 87] Clocksin W. F. e Mellish C. S.
Programming in Prolog,
Springer-Verlag.

[Banerjee 87] Banerjee J. e Chou H. T.,
Data Model Issues for Object-Oriented Applications,
ACM on Office Information Systems,
Vol. 5, No. 1, pg 3 - 26,
New York, janeiro 1987.

[Souza 88a] Souza L. F. P. de e Dallolio V. M.,
Implementação de Corte e Disjunção na Máquina Abstrata Prolog,
I Coletânea de Resultados de Pesquisa, Projeto Estra, SID Informática,
pg 231-238,
Brasília, DF, Abril de 1988.

[Schmitz 88] Schmitz E. A., Teles A. A. S. e
Azevedo M. H. C de,
PI - Um Processador de Inferências,
Anais do 8º SBC, SBC/UFRJ,
pg 14-26,
Rio de Janeiro, RJ, julho de 1988.

[Souza 88b] Souza L. F. P. de e Dallolio V. M.,
Pseudo-instruções Prolog e seu Ambiente de Execução,
Relatório Técnico NCE-12/88, NCE/UFRJ,
Rio de Janeiro, RJ, outubro de 1988.

[Dallolio 88] Dallolio V. M. e Souza L. F. P. de,
Geração de Código para Cláusulas Disjuntivas na Máquina Abstrata Prolog,
Anais de 5º SBIA, SBCIA/UFRN,
pg 541-550,
Natal, RN, novembro de 1988.

APÊNDICE - Tabela de Operadores

OPERADOR	ASSOCIATIVIDADE	PRECEDÊNCIA
:-	xfx	1200
:-	fx	1200
-->	xfx	1200
?-	fx	1200
;	xfy	1100
,	xfy	1000
nospy	fy	900
spy	fy	900
not	fy	900
\+	fy	900
->	xfx	800
=	xfx	700
is	xfx	700
=..	xfx	700
\=	xfx	700
@<	xfx	700
@=<	xfx	700
@>	xfx	700
@>=	xfx	700
=\=	xfx	700
<	xfx	700

(Continua na próxima página)

OPERADOR	ASSOCIATIVIDADE	PRECEDÊNCIA
>	xfx	700
=<	xfx	700
>=	xfx	700
==	xfx	700
\==	xfx	700
=:=	xfx	700
/\	yfx	500
\/	yfx	500
:	fy	500
+	yfx	500
-	yfx	500
+	fx	500
-	fx	500
*	yfx	400
/	yfx	400
>>	yfx	400
<<	yfx	400
//	yfx	400
mod	xfx	300
^	xfx	200